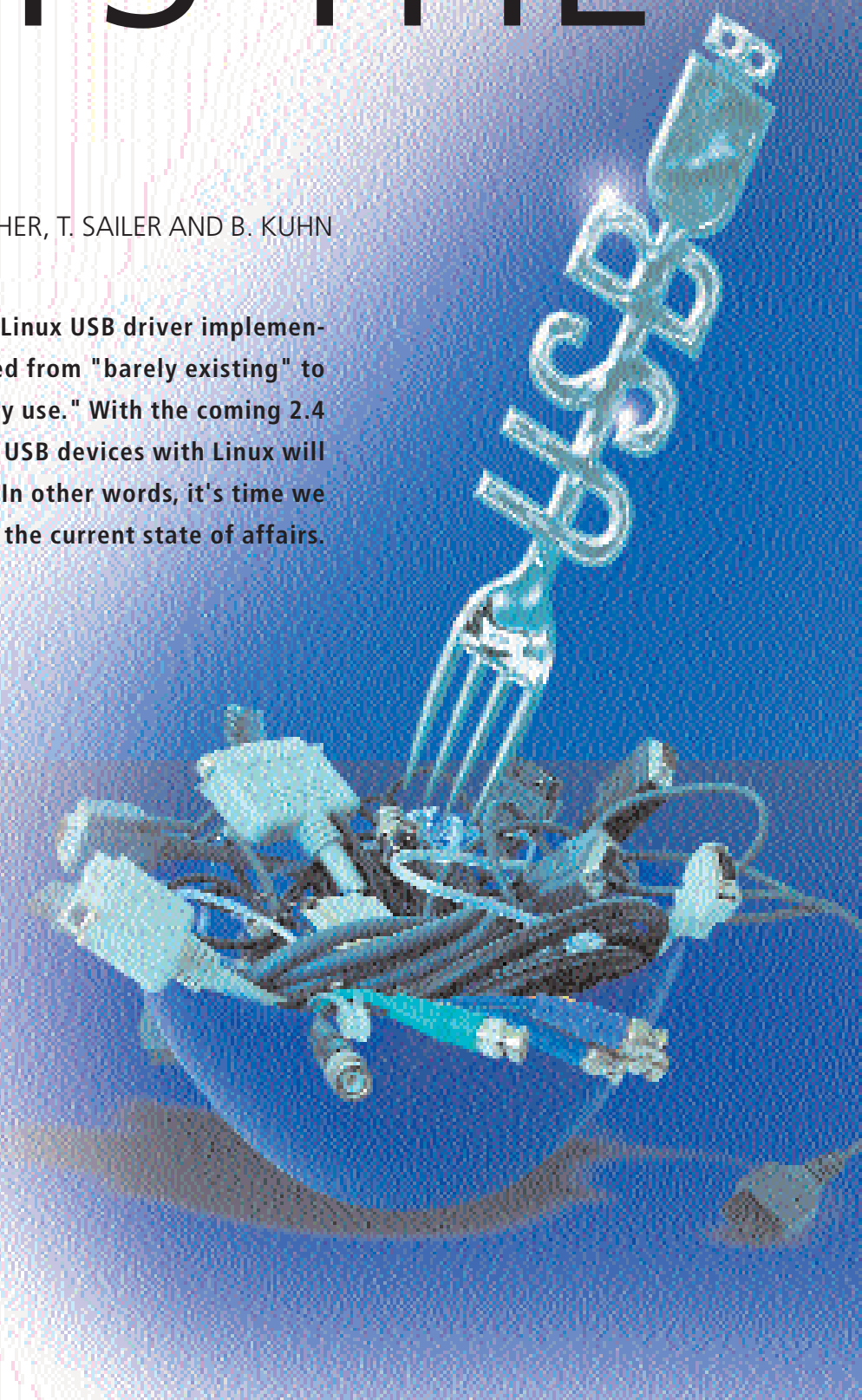
Universal Serial Bus

LINUX GETS THE BUS

D. FLIEGL, G. ACHER, T. SAILER AND B. KUHN

Within a short time, the Linux USB driver implementation has blossomed from "barely existing" to "suitable for everyday use." With the coming 2.4 kernel series the use of USB devices with Linux will be regarded as standard. In other words, it's time we took a look at the current state of affairs.



Since 1994, the "Universal Serial Bus" has been promoted by its initiators (Compaq, Intel, Microsoft and NEC) as the new standard interface for peripherals requiring low to medium data rates. However, the developers of peripheral drivers frequently had only a single target operating system family in their sights. Consequently for Linux the support remained nothing more than an interface for what was regarded up to that point as the "Useless Serial Bus."

The multimedia USB architecture enabled other developers to quickly develop drivers for the widely available and attractively priced USB devices, as long as the hardware manufacturers followed the USB 1.1 specification or were prepared to hand out data sheets and manuals.

Originally USB support was only included in the experimental developer series 2.3.*/2.4.0-test*. However, in the meantime there was a well-maintained "Backport Patch" for Kernel 2.2.16. Impatient distributors have (at the request of impatient users!) already included the driver software in their current distributions. Thus SuSE, Mandrake and Caldera already offer USB support "out-of-the-box."

Sadly, the support offered by the distributor rarely goes beyond configuration of the USB mouse

USB Principles

The "Universial Serial Bus" has a strictly hierarchical structure and is managed by a host controller. The host uses a master/slave protocol to communicate with the USB devices that are connected: in other words all data transfers are initiated by the host. USB devices cannot communicate with each other, which is of no great significance to peripheral equipment and saves on costs, as the hardware and software need no great intelligence. Also, this means that problems like collision detection or bus arbitration don't arise. At the moment, a USB enables a maximum of 127 devices to be connected, providing a bandwidth of 12MBit/s via the four-pole connecting cable (+5V, ground, data+ and data-), but this can only be used to 90 per cent capacity. In the case of the USB 2.0 specification recently shown for the first time, up to 480MBit/s should be possible. To enable the synchronisation of multimedia dataflow, such as audio or video, USB transactions are embedded in a frame structure. One frame lasts exactly 1 ms (12,000 bits).

Normally, the required USB host controller is integrated into the motherboard. Older circuit boards can be retrofitted with suitable PCI cards. In spite of the multitude of USB chip sets in existence, the manufacturers fortunately stick to two standards only: the "Universal Host Controller Interface" (UHCI) developed by Intel and the "Open Host Controller Interface" (OHCI) from Compaq and Microsoft. This makes no difference for USB device drivers (rather as in the case of SCSI host adapters.)

What are known as "class specifications" exist for frequently used USB devices such as modems, bulk storage, keyboards, mice, joysticks, monitors, audio devices, printers and USB-to-IrDA converters. Therefore, it should be the case that as many devices as possible with the same functions will work with the same driver. Unfortunately, there are no open class specifications in existence for webcams, digital cameras, scanners and USB/RS232 converters and a special driver is needed as a rule. In the case of Linux this causes familiar problems.

Although the programming models of typical USB devices certainly don't display any technical peculiarities some manufacturers are trying not to give out specifications or will do so only if a non disclosure agreement is signed. Therefore, the development of drivers for these devices is difficult or even downright impossible and at best only possible with time-consuming reverse engineering (with the help of tools like "USB-Snoopy" and "Playback" – see "Info" box.)

The functions of USB devices are logically subdivided into so-called "interfaces". An interface covers all communication with a particular part of the device. An interface can have different operating modes, known as "Alternate Settings". Only one alternate setting can be active at one time. Audio input and output interfaces, for instance, use alternate settings in order to distinguish different scanning formats. An interface can include several end points. An end point is to a certain extent comparable with a TCP/IP port. End points are, however, unidirectional, which means that data can only be transferred in one direction.

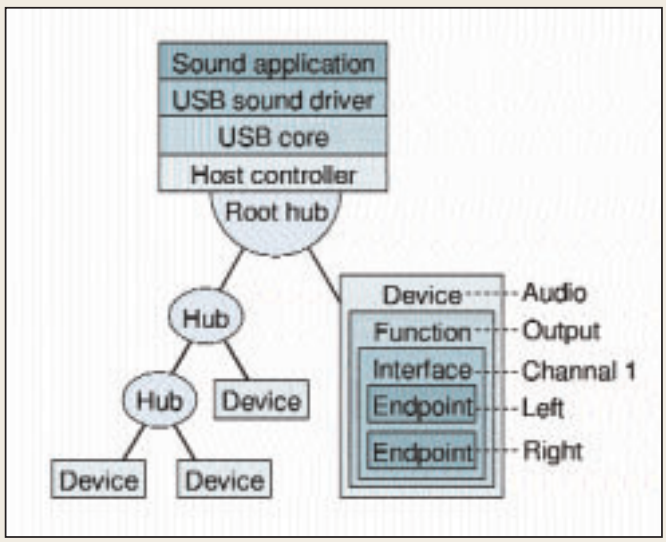


Fig. 1: The world turned upside down. In reality, the universal serial bus has a star structure and the hubs work physically like switches. The drivers communicate with one another using the end points of the interfaces of a device.

and keyboard. As before, knowledge of the Universal Serial Bus and the Linux USB driver architecture is necessary when operating scanners, printers and multimedia devices, but as a user of the one of the abovementioned distributions you do at least save yourself the kernel installation procedure described in the box "USB installation and configuration" as all the necessary driver modules will have been "factory compiled".

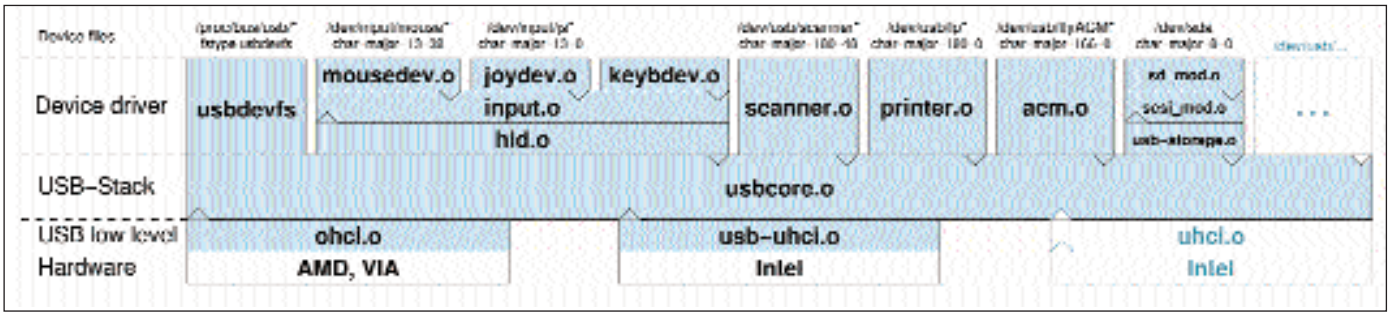
Kernel modules

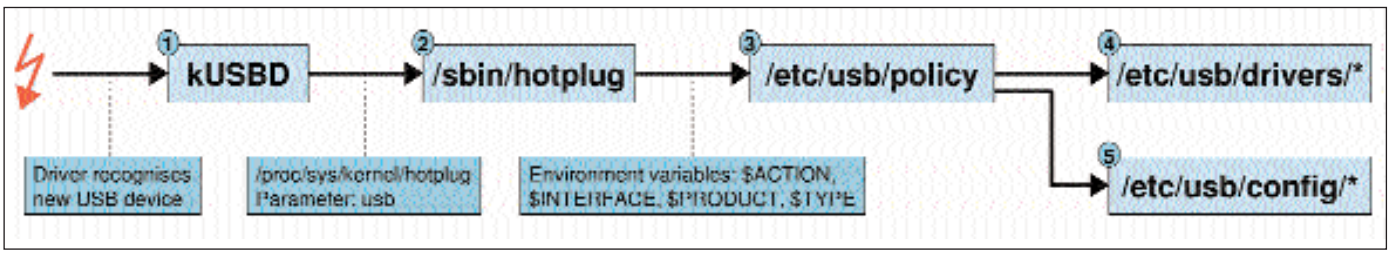
The structure and dependencies of the device drivers for the Linux USB system are shown in Figure 2. The basis for all USB drivers is the combination of the USB core system (*usbcore*) with at least one host controller driver (*usb-uhci* or *usb-ohci*). Apart from the integrated hub driver, the core driver offers the option of creating an information structure in the */proc* directory ("USB device file system".) It is with this in user mode that drivers can also access USB devices. The *usbdevfs* also enables tools such as *lsusb* to output a list of connected devices (analogous to *lspci* for plug-in cards in a PCI bus).

After compilation and installation of the modules the drivers can be integrated as usual into the current kernel using *insmod*:

```
insmod usbcore
insmod usb-uhci # or: insmod usb-ohci
```

You can check the success of this action with *dmesg*. If no USB device is connected, at least the host controller should signal and output the number of USB ports detected. Further hubs are also





detected by the USB core driver and automatically initialised, which in the case of most hubs is signalled by the lighting up of one or more LEDs. Connected or newly plugged-in end devices are also noticed by the USB core driver, but cannot be used without a special device driver (*usb-storage*, *joydev/input/hid* etc.).

In the case of many distributed products with USB support built in at the factory (SuSE, Mandrake and Caldera), a simple script run at system boot-up takes care of the integration of device-dependent driver modules. As a rule, only mouse and keyboard driver modules are loaded regardless. The modules for printers, joysticks, etc. added later must be loaded manually – as long as suitable entries are not present in */etc/modules.conf*. For example, for a 3 1/2" USB disk drive, the entry might look like this:

```
alias scsi_hostadapter usb-storage
```

You can now address the disk drive via */dev/sda* (as long as the appropriate SCSI modules have been compiled, of course). If you also adapt the entry in

[top]
Fig. 2: Linux USB implementation is greatly modularised.

[above]
Fig. 3: Connecting new USB devices causes a chain reaction.

Audio, Communication, Joysticks, Hubs and Mice

Vendor	Product	Comment
Audio equipment		
Philips	PCA 646BC	Microphone for Webcam
Canopus	Canopus DA-Port USB	D/A converter
Dallas Semiconductor	USB DAC	Loudspeaker
Roland	MA-150U	Loudspeaker
Communication equipment		
Vendor	Product Comment	
3Com	OfficeConnect Analogue Modem	56k Business Modem
3Com	U.S. Robotics ISDN Pro TA	ISDN Terminal Adapter
3Com	U.S. Robotics Model 5605	56k Voice Fax-modem
Compaq	USB Modem	ACM modem
Diamond	Diamond Supramaz 56k Usb (2890)	Atlas Modem Board (Lucent Technologies)
Diamond	Supraexpress 56k USB Modem	SUP2780
Digicom	Tintoretto USB	Modem ISDN USB
Elsa	Microlink 56K USB	V.90 Modem
Entrega Technologies	Hub3U1E	4 -Port-Hub / Ethernet
Linksys	USB100TX	10/100 USB-10baseT Ethernet Adapter
Lucent Technologies	CNet SinglePoint 56Kbps V.90 Fax Modem	Atlas Modem Board
MELCO	LUA-TX	USB 10/100M LAN Adapter
Metricom	Ricochet G2	wireless modem
Netgear/LiteOn	ea101c/LNE100TX	USB to Ethernet Adapter
OXUS RESEARCH	OXUS-B ISDN	ISDN Modem
Sirius Technologies	NetComm Roadster II 56 USB	56K Flex/V90 Modem
Telecom Device	TCD-UFE100	USB 10/100M LAN Adapter
Zoom Telephonics	2986L	V90 Fax-modem
Joysticks and Game Pads		
Vendor	Product Comment	
CH Products	CH 3-Axis 10-Button+POV USB Joystick	F-16 Combat Stick
CH Products	CH Pro Pedals USB	aircraft pedals
CH Products	FlightSim Yoke LE	aircraft control lever
Gravis	Game Pad Pro USB, Model #4211	Digital Joystick
Logitech	WingMan Extreme Digital 3D	6-axis, 7-button USB/Gameport Joystick
Logitech	Wingman Game Pad	USB HID 2-axis, 11-button game pad
Microsoft	Microsoft SideWinder Plug & Play Game Pad	Microsoft SideWinder Game Pad (2 axes, 6 buttons)
Microsoft	Sidewinder Game Pad Pro	USB HID (2 axes, 6 buttons)
Microsoft	Sidewinder Precision Pro	6 axes, 9 buttons
Rockfire	RM-203u (USB-Nest)	USB/Gameport converter

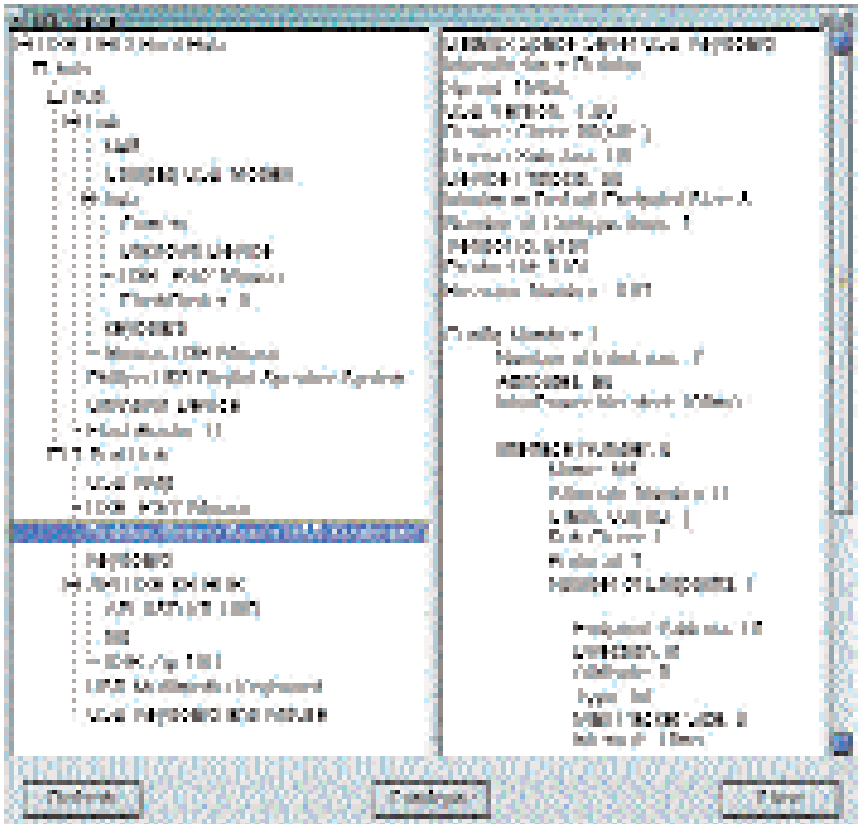


Fig. 4: There are also colourful tools for Linux to display the USB devices currently connected.

/etc/mtools.conf (replace "fd0" with "sda" and delete "1.44m"), the USB drive can be accessed using *mtools*. However, the driver is not all that stable for bulk storage yet (something we were able to confirm using a Sony Vaio N505X notebook.)

Agents and policy

Sadly, the loading of device-dependent USB modules cannot be fully automated with */etc/modules.conf*, but having to load a dozen drivers in advance is not what the inventor was thinking of nor is it particularly efficient. Consequently, in the 2.4 version of the kernel there is a module loading mechanism for "hot pluggable devices" such as PC cards, USB and CPCI. With:

```
echo "/sbin/hotplug" > /proc/sys/kernel/hotplug
```

you can inform the "hardware agents" which command should be called after detection of new hardware. */sbin/hotplug* is included with the "usbd scripts" package, which you simply unpack to */etc/usb* after the download. *hotplug* must be moved to */sbin* and, as shown in Figure 3, calls */etc/usb/policy* as soon as a USB device has been inserted or removed. It is only decided at this level which driver is to be loaded and whether or not additional configuration measures are to be taken.

The README of the "usbd scripts" package requests that the /etc/usb/rc.usb be accepted into the boot mechanism. In the case of Red Hat for example in /etc/rc.d/rc.sysinit after activation of the virtual memory (swapon -a >/dev/null 2>&1) by means of:

```
# Set up USB
if [ -x /etc/usb/rc.usb ]; then
  action "Starting USB" /etc/usb/rc.usb start
fi
```

The *rc.usb* script mounts */proc/bus/usb* for system programs such as *usbview* (see Figure 4) and loads all USB modules specified in */etc/sysconfig/usb*. Anyone wanting to fully automate the subsequent loading of modules must add the devices to the system using the manufacturer ID, the device code and possibly its version number, if necessary. The "Red Shooter USB Joystick", for example, has the manufacturer number 663 and device number 9805 (the version number is irrelevant here). Otherwise, the

Troubleshooting

The most frequent cause of non-functioning USB devices are missing drivers or missing/incorrect entries in the /dev directory. Missing drivers are detected by the USB core driver and noted in the kernel log, so a check can be made at least to see whether the device is supported by a driver that has been loaded. Given the rapid rate of Linux USB development it can also happen that possible existing /dev entries have the wrong major/minor number. This can be clarified by taking a glance at the relevant documentation (/usr/src/linux/Documentation/devices.txt).

As a result of the virtually unlimited possible combinations of host controller and USB devices, occasional hardware problems can occur, such as communication errors during the detection process. In such cases, the USB mailing list archive offers further help.

Many USB devices and above all hubs can be pretty sensitive to electrical disturbance such as may occur when plugging in or switching on main adapters or fluorescent lamps. Normally, this leads to the switching off of the hub and the "loss" of all the USB devices that are connected to it. The Linux hub driver, as opposed to Windows, is so smart that it notices this and reactivates the hub (message: "already running port %i disabled by hub (EMI?), re-enabling..."). If the individual USB drivers support reconnection (such as the mouse driver, for instance), the USB device will only be incapable of being addressed for a few seconds. It is possible, however, that the associated user program will still have to be restarted.

code numbers can be determined using *dmesg*, *lsusb* or *usbview*. The appropriate entry in */etc/usb/drivers/hid* will then look like this:

```
663/9805/*)
  if $MODPROBE joydev >/dev/null 2>&1
  then
    message ... loaded joydev
    DRIVER=hid
  fi ;;
```

Devices supported

At the moment, a large number of USB devices are already supported by Linux because an astonishing number of manufacturers are abiding by the predefined class specifications. But as usual, exceptions confirm the rule. Before acquiring a USB device it is worthwhile taking a look at the constantly growing list of devices supported. Tables 1 to 3 give a brief overview, but make no claims to being complete: in particular the USB mice and keyboards not listed here shouldn't present any problems with Linux.

A tip when using USB mice with X: instead of specifying a device file of a specific mouse (e.g. */dev/input/mouse0*) in the file *XF86Config* it is better to use */dev/input/mice*. USB mice will then let you plug them in and out during operation without having to restart the X server.

Worth mentioning as an especially exotic USB device at this point is the Prolific PL-2302 USB-to-

USB installation and configuration

Before putting USB devices into operation under Linux you must first surmount the obstacle of kernel compilation. To do this you need up-to-date kernel sources and the backport patch for the 2.2 kernel series. In the case of the 2.4 test series the USB drivers are already included, but very few distributed products are adapted to the new kernel or even offer it as an option.

```
cd /tmp
wget -c ftp://ftp.de.kernel.org/pub/linux/kernel/v2.2/linux-2.2.16.tar.bz2
wget -c http://www.suse.cz/development/usb-backport/usb-2.4.0-test2-pre2-for-2.2.16-v3.diff.gz
cd /usr/src
mv linux linux.old
bunzip2 -cd /tmp/linux-2.2.16.tar.bz2 | tar xf -
cd linux
gunzip -cd /tmp/usb-2.4.0-test2-pre2-for-2.2.16-v3.diff.gz | patch -p1
make menuconfig
```

Apart from the usual options, the kernel USB configuration menu offers quite a few selection possibilities. The easiest is to activate all the USB drivers as a module: any drivers definitely not needed can be left out and compiled later if necessary. However, if you have a USB keyboard the "Support for USB" (usbcore), the driver for the host controller (UHCI or OHCI), as well as "Keyboard Support" should be permanently included in the kernel compilation, otherwise it is only with tricks (e.g. with the "Initial Ramdisk") that you can guarantee the keyboard will work in critical exceptional cases.

In the case of other kernel menu options a look at the default settings of the kernel configuration or the distributor's manual will help. After a:

```
make clean && make dep && make bzImage
make modules && make modules_install
```

the freshly baked kernel must be transferred as usual to /boot and integrated into the boot process using LILO, Loadlin, Grub or similar mechanisms.

Two entries are still missing at the start of the file /etc/conf.modules (or /etc/modules.conf in the case of newer modutils):

```
keep
path[usb]=/lib/modules/`uname -r`
```

The first line tells the commands insmod, modprobe and depmod to maintain the existing path list and to add the following path configuration to the existing list. In our example the existing path is simply extended to the directory that has the modules for USB devices.

The following entry in /etc/fstab ensures that the USB kernel interface will be accessible for various USB system programs after the next reboot:

```
none /proc/bus/usb  usbdevfs  defaults
```

Alternatively, the USB device file system can be integrated manually:

```
mount -t usbdevfs none /proc/bus/usb
```

Supported mass storage and serial interfaces			
Vendor	**Product Comment**		
Mass storage devices			
AIWA	TD-U8000	Tape Drive	
Caravelle	RW-448USB	CD-RW	o
Castlewood	ORB2SE00	2.2GB Removable Media Hard Drive	o
Hagiwara Sys-Com	FlashGate CF	CompactFlash Reader/Writer	o
Hagiwara Sys-Com	FlashGate DUAL	PC-Card & SmartMedia Reader/Writer	o
Iomega	USB Zip 250	250 MByte	o
Iomega	Zip Drive USB	100 MByte	
LaCie	Hard Disk		o
Microtech International Inc	USB-SCSI-DB25	OEM Shuttle SCSI Controller.	-
SanDisk Corporation	ImageMate CompactFlash USB	Compact Flash Reader SDDR-31	
Shuttle Technologies, Inc.	HP USB CD-Writer Plus		o
Sony	MSAC-US1	Memorystick Adapter	o
Sony	Spressa USB Plus (CRX100E/X2)	CD-RW 4x/4x/6x	o
VIPowER	USB MobileRACK	USB/IDE Adapter	
Y-E DATA	FlashBuster-U	3 .5-inch Floppy	
Serial interfaces			
Vendor	Product Comment		
ConnectTech	WhiteHEAT	4 x Serial	o
Digi International	Acceleport USB 4	4 x Serial	o
HandSpring	Visor	Serial Emulation	
Keyspan	PDA Adapter	1 x Serial	o
Keyspan	USA-19	1 x DB9, 57600 bit/s	o
Keyspan	USA-19W	1 x DB-9, 230400 bit/s	o
Keyspan	USA-28	2 x Din-8, 115200 bit/s	o
Keyspan	USA-28X	2 x Din-8, 230400 bit/s	o

Webcams, Digital Cameras, Keyboards, Printers	
Vendor	**Product**
Webcams	
Philips	PCA 646VC
Askey	VC010 (Type 1 & 2)
Askey	VC080 (CMOS Sensor)
AvCam	AvCam 600 USB
Creative Labs	Webcam 3 (CT6840)
Lifeview	Robocam
Maxxtro	PC Camera (OV511-based)
Mustek	VCAM-300
Philips	PCA645VC, 645VC
Philips	PCVC675K, 680K
Terratec	USB Camera (CPiA-based)
Tevion	Model 9308 (ALDI-Webcam)
Trust	Spacec@m Lite
Zoom Telephonics	ZoomCam USB 1595
Digital Cameras	
Vendor	Product
Canon	Powershot S10
Kodak	DC 220, 240, 260, 265, 280, 290
Mustek	MDC 800
RICOH	RDC-5000
Sony	DSC-070, DSC-F505, DSC-F505V
Toshiba	PDR-M4, M5
Printers	
Vendor	Product
Aten	UC-1284B Printer Cable Adapter
Brother	HL1250
Canon	BJ F300, NJC-3000
Epson	Stylus Color 670, 640, 760
Epson	Stylus Photo 1270
Hewlett-Packard	DeskJet 810C, 812C, 840C 880C,
Hewlett-Packard	DeskJet 895C, 970Cse, 1220C
Hewlett-Packard	Photosmart P1100
Lexmark	Optra S 2450, E 310
Scanners	
Vendor	Product
Microtek	X6USB
AGFA	Snapscan1212u_2
Acer	Brisa 640U
Colorado USB	9600
Epson	Perfection 1200 U / P
Hewlett-Packard	Scanjet 4100C, 5200, 5200C
Hewlett-Packard	Scanjet 6200C, 6300C
Umax	Astra 1220U

USB adapter. This is a type of null modem cable for connecting two computers via USB. The driver installs itself as a network device (*plusb**). You can then set up a connection with *ifconfig plusb0 10.0.0.1 pointopoint 10.0.0.2*. The transfer rate is approximately 5 MBit/s.

Programming with URBs

Now we'll take a brief look at USB driver development. The basis of communications is the so-called "USB Request Block" (URB). The URB design is orientated towards the Windows API, but of course does not use Windows code and has been improved in many respects and made compatible with the Linux environment.

A URB is a data structure which transfers to the USB subsystem all the information necessary for a transfer (device, interface, end point, user data). The transfer specified in the URB is started by the host controller driver and runs in the background until it is finished. In other words, after the URB has been sent the caller doesn't have to wait until the end of the transaction but can carry on with other work at the same time (asynchronous behaviour). Once a transfer has been finished later, either successfully or with errors, the system skips to the callback function ("Completion routine") specified in the URB, which can then further process the data and also trigger new transfers.

With the aid of the kernel function *usb_alloc_urb()*, URBs can be allocated and later linked so that the successful execution of a USB request automatically triggers the next one. This is especially useful for continuous data streams as the completion routine only has to deliver or retrieve the data (*usb_submit_urb()*) – everything else takes

place automatically in the USB subsystem. If a USB request runs into a timeout, the associated URB can be deleted (*usb_unlink_urb()*).

However, for many USB devices it is no longer necessary to program a "real" kernel driver as almost all functions can be addressed when in user mode by means of the USB device file system (usbdevfs), thereby considerably simplifying driver development.

Integrating USB drivers

Apart from the normal driver entry points (*open()* etc.), each USB driver module has two further functions, which are called on plugging in (probing) or disconnecting a device. During its initialisation with *usb_register(struct usb_driver*)*, a USB device driver must preferably register in the area reserved for USB drivers (major-180) – the minor start value is defined in the *usb_driver* structure.

For each new device plugged in and its interface, the probe functions of all currently loaded USB kernel drivers are called with a pointer to the device structure and the interface number, in the hope that a driver accepts this interface. In this case it returns the value !=ZERO and is thus connected to this device and interface. Probing takes place for each interface individually, so that multifunctional devices can also be recognised and used. With "hot-plug support" activated (in the 2.4 kernel only) it is possible to carry out further actions with */proc/sys/kernel/hotplug* (see Figure 3).

On unplugging the device, the disconnect function is called. This is a critical point, because the associated device file could still be opened by an application. Further accesses must result in an error message instead of letting the kernel driver crash with zero pointers.

USB driver in user mode

The USB device file system offers an extensive range of options for addressing USB devices when in user mode, without requiring a special kernel driver. The *usbdevfs* is usually mounted on */proc/bus/usb*.

- **/proc/bus/usb/devices** This file lists all the USB devices connected and their characteristics. A *select(input)* on this file triggers the application called as soon as a USB device is plugged in or removed.
- **/proc/bus/usb/drivers** This file lists all currently loaded kernel USB device drivers.
- **/proc/bus/usb/ bus number>/ device adresse>** Using this file, transfers can be initiated to and from USB devices.

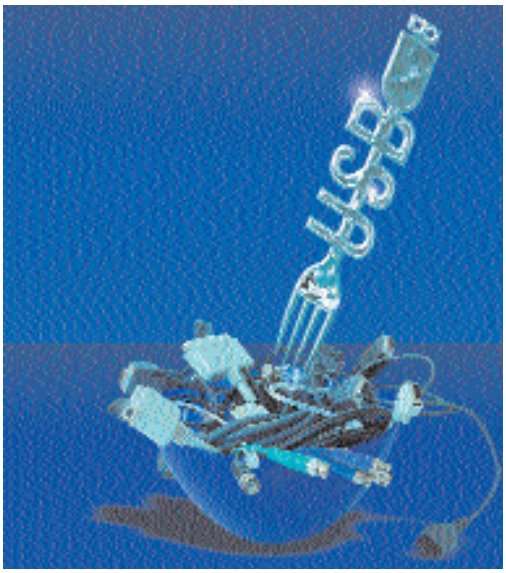
The API is heavily modelled on the kernel URB API. Instead of calling kernel functions such as *submit_urb()*, *ioctl* calls arise and the parameters are transferred as pointers to a structure. A good, simple example which illustrates this mechanism for all types of transfer is the "usb stress module".

Developing your own USB hardware

If you wish to develop your own USB devices the 8051 derivatives by AnchorChips are very useful. (They were also used, for instance, in the Digital Audio Broadcast USB project.) These modules can easily be programmed using a USB interface. An example of this can be foundin the "usbstress" package.

USB in the future

USB for Linux has emerged from the hacker phase and already provides kernel support for many common USB devices. As a result of the very stable URB API a new driver comes into being virtually every week. However, putting USB peripherals into operation under Linux is still not so easy and the user may well need further assistance. ■



Info

** Universal Serial Bus Specification, Revision 1.1 (September 1998), USB Implementer's Forum*
** Open Host Controller interface (OHCI) Specification, 1996, Compaq*
** Universal Serial Bus Device Class Definition for Audio Devices (Release 1.0, March 1998), USB Implementer's Forum*
** Universal Host Controller interface (UHCI) Design Guide, Revision 1.1, March 1996, Intel Corp.*
***"USB-Snoopy"**: http://www.jps.net/~komal/*
***"Playback"**: http://usb-robot.sourceforge.net/*
***lsusb and libusb**: http://usb.in.tum.de/download/usbutils/usbutils-0.6.tar.gz*
***List of USB devices supported:** http://www.qbik.ch/usb/devices/*
***Mailing list archive of linux-usb@suse.com**: http://electricrain.com/lists/archive/linux-usb*
***8051 microprocessors with USB:** http://www.anchorchips.com*
***Homepage of the Linux USB project:** http://usb.in.tum.de*
***Linux USB Guide:** http://linuxusbguide.sourceforge.net/*
***Homepage of the USB for Linux project** http://www.linux-usb.org*

■